AMENDMENTS TO THE CLAIMS

1. (Currently amended) A method of verifying program code conversion performed by an emulator, comprising the steps of:

a) dividing a subject code into a plurality of blocks and executing one of the blocks of subject code through an emulator in a process image on a subject processor according to an emulation context up until a comparable point in the subject code to provide an emulated machine state;

b) performing a context switch to a native context and executing the same block of subject code natively in the same process image on the same subject processor up until the same comparable point in the subject code to provide a native machine state; and

c) comparing the native machine state from execution of the one block of subject code natively on the subject processor against the emulated machine state from execution of the same block of subject code on the same subject processor through the emulator at the comparable point in the subject code;

wherein the native machine state includes a native memory image and the emulated machine state includes an emulated memory image and the step (a) and/or the step (b) includes selectively isolating access to a memory associated with the subject processor to obtain the native memory image and/or the emulated memory image, respectively.

2. (Cancelled)

3. (Currently amended) The method of claim [[2]]1, comprising performing the step (a) prior to performing the step (b).

4. (Previously presented) The method of claim 3, wherein:

the step (a) further comprises providing an emulated image of the subject processor;

the step (b) further comprises providing a native image of the subject processor following the native execution of the program code; and

the step (c) further comprises comparing the emulated image of the subject processor against the native image of the subject processor.

5. (Original) The method of claim 4, wherein the step (a) comprises providing the emulated image of the memory in a load/store buffer associated with the memory, such that the memory is not affected by executing the subject code through the emulator.

6. (Original) The method of claim 5, wherein the emulated image of the subject processor includes an image of one or more registers.

7. (Original) The method of claim 6, wherein the emulated image of the subject processor includes an image of one or more condition code flags.

8. (Cancelled)

9. (Cancelled)

10. (Previously presented) The method of claim 1, comprising selectively switching between the emulation context for running the emulator on the subject processor, a target execution context for executing target code produced by the emulator on the subject processor, and the native context where the subject code runs natively in the subject processor.

11. (Original) The method of claim 10, wherein both the native context and the emulation context employ a single image of the subject code.

12. (Cancelled)

13. (Previously presented) The method of claim 1, comprising selecting between two or more verification modes, and dividing the subject code into the plurality of blocks according to the selected verification mode.

14. (Previously presented) The method of claim 13, comprising repeating the executing and comparing steps for each of the plurality of blocks.

15. (Previously presented) The method of claim 14, wherein in a first verification mode each block comprises a single instruction of subject code; in a second verification mode each block comprises a basic block comprising a sequence of instructions from a unique entry instruction to a unique exit instruction; and in a third verification mode each block comprises a group block comprising a plurality of the basic blocks.

16. (Original) The method of claim 1, comprising the steps of:

dividing a large segment of the subject code into a plurality of smaller blocks, each block containing one or more instructions from the large segment of subject code; and

performing a verification comparison at a block boundary between each pair of consecutive neighbouring blocks in the plurality of blocks.

17. (Previously presented) The method of claim 16, comprising the steps of:

providing the subject processor in the emulation context, where control of the processor rests with the emulator, performing program code conversion on a current block BBn to produce a corresponding block of converted target code, and patching an immediately preceding block of subject code BBn-1 with a return jump;

executing a context switch routine to enter the native context, and executing the immediately preceding block of subject code BBn-1 natively by the subject processor, such that the executing step terminates with the return jump;

executing a context switch routine to return to the emulation context, and performing the verification comparison by comparing a native machine state representing the subject processor following execution of the immediately preceding block BBn-1 with an emulated machine state representing a virtual model of the subject processor held by the emulator following execution of the immediately preceding block BBn-1;

executing a context switch to a target execution context, and modelling execution of the target code corresponding to the current block of subject code BBn in the virtual model of the subject processor held by the emulator, thereby leaving the virtual model in a machine state representing the end of the current block BBn; and

repeating the above steps for each subsequent block in the plurality of blocks, unless the verification comparison reveals an error in the program code conversion.

18. (Original) The method of claim 17, further comprising restoring the immediately preceding block BBn-1 to remove the return jump.

19. (Original) The method of claim 1, further comprising the steps of:

selecting a block of the subject code;

executing the block of subject code on the subject processor through the emulator; and

appending a return jump to the block of subject code, and executing the block of subject code natively on the subject processor terminating with the return jump, such that the return jump returns control of the processor to the emulator.

20. (Currently amended) A method of verifying program code conversion, comprising the steps of:

dividing a subject code into a plurality of blocks and performing program code conversion to convert one of the plurality of blocks of subject code into target code through an emulator running in a process image on a subject processor according to an emulation context and executing the target code to provide an emulated machine state including selectively inhibiting access by the emulator to a memory associated with the subject processor by buffering load and store requests from the subject processor to the memory in a load/store buffer;

switching to a native context and executing the same one block of subject code directly in the same process image on the same subject processor to provide a native machine state that is stored in the memory associated with the same subject processor; and

comparing the emulated machine state contained in the load/store buffer against the native machine state contained in the memory to verify the program code conversion.

21. (Cancelled)

22. (Previously presented) The method of claim 20, comprising selectively inhibiting access to the memory when executing the target code, such that an emulated memory image is provided in the load/store buffer.

23. (Previously presented) The method of claim 20, wherein after the program code conversion performed by the emulator running on the subject processor has been verified, the method further comprising the step of:

comparing execution of the subject code through the emulator running on the subject processor against execution of the subject code through a second emulator running on a target processor.

24. (Original) The method of claim 23, comprising providing a first host processor as the subject processor, and providing a second host processor as the target processor.

25. (Original) The method of claim 24, wherein the subject code is natively executable on the subject processor whilst not being natively executable on the target processor.

26. (Currently amended) A method of verifying program code conversion, comprising the steps of:

first dividing a subject code into a plurality of blocks and comparing execution of one subject code block natively on a subject processor against execution of the same subject code block on the same subject processor through a first emulator both in a single process image, thereby verifying program code conversion performed by the first emulator; and

also comparing execution of same subject code block through the first emulator running on the subject processor against execution of the same subject code block through a second emulator running on a target processor, thereby verifying program code conversion performed by the second emulator using the verified program code conversion performed by the first emulator.

27. (Previously presented) The method of claim 26, comprising the steps of:

performing a first program code conversion of the subject code including providing a first virtual model of the subject processor in the first emulator, and comparing the first virtual model against the subject processor; and

performing a second program code conversion of the subject code including providing a second virtual model of the subject processor in the second emulator, and comparing the first virtual model in the first emulator against the second virtual model in the second emulator.

28. (Original) The method of claim 27, comprising providing a single way communication from the first emulator to the second emulator.

29. (Previously presented) The method of claim 27, comprising the steps of:

synchronising the first and second virtual models by sending initial state information from the first emulator to the second emulator;

for each block of subject code, executing the block of subject code through the first emulator and providing a set of subject machine state data and non-deterministic values to the second emulator;

executing the block of subject code in the second emulator substituting the non-deterministic values and providing a set of target machine state data; and

comparing the subject machine state data against the target machine state data and reporting an error if a divergence is detected, otherwise repeating the process for a next block of subject code.

30. (Currently amended) A method of verifying program code conversion, comprising the steps of:

(a) dividing a subject code into a plurality of blocks, wherein each block includes at least one instruction;

(b) executing one of the blocks of subject code in a process image on a subject processor through a first emulator according to an emulation context;

(c) switching to a native context, executing the one block of subject code natively on the same subject processor in the same process image and comparing the native execution against the execution of the same one block of subject code on the same subject processor through the first emulator, thereby verifying program code conversion of the block of subject code performed by the first emulator;

(d) comparing execution of the same one block of subject code through a second emulator running on a target processor against the already verified execution of the same one block of subject code through the first emulator running on the subject processor, thereby verifying program code conversion of the one block of subject code performed by the second emulator; and

(e) repeating steps (b)-(d) for each of the plurality of blocks of the subject code until program code conversion performed by the second emulator is verified for each of the plurality of blocks of the subject code.

31. (Previously presented) The method of claim 30, wherein the dividing step comprises dividing the subject code such that each of the plurality of blocks of subject code contains a single instruction.

32. (Previously presented) The method of claim 31, wherein after program code conversion performed by the second emulator is verified for each of the plurality of blocks of subject code containing a single instruction, the method further comprises:

secondly repeating the step (a) by redividing the subject code into a plurality of new blocks, wherein each new block is a basic block comprising a sequence of instructions from a unique entry instruction to a unique exit instruction; and

repeating steps (b)-(e) for each basic block, thereby verifying program code conversion performed by the second emulator for every basic block of subject code.

33. (Previously presented) The method of claim 32, wherein after program code conversion performed by the second emulator is verified for each of the basic blocks of subject code, the method further comprises:

thirdly repeating the step (a) by redividing the subject code into a plurality of group blocks, wherein each group block comprises a plurality of basic blocks; and

repeating the steps (b)-(e) for each group block, thereby verifying program code conversion performed by the second emulator for every group block of subject code.

34. (Currently amended) A computer-readable storage medium having emulator software resident thereon in the form of computer readable code executable by a computer to verify program code conversion performed by an emulator by performing the steps of:

a) dividing a subject code into a plurality of blocks and executing one of the blocks of subject code through an emulator in a process image on a subject processor according to an emulation context up until a comparable point in the subject code to provide an emulated machine state;

b) performing a context switch to a native context and executing the same block of subject code natively in the same process image on the same subject processor up until the same comparable point in the subject code to provide a native machine state; and

c) comparing the native machine state from execution of the one block of subject code natively on the subject processor against the emulated machine state from execution of the same block of subject code on the same subject processor through the emulator at the comparable point in the subject code;

wherein the native machine state includes a native memory image and the emulated machine state includes an emulated memory image and the step (a) and/or the step (b) includes selectively isolating access to a memory associated with the subject processor to obtain the native memory image and/or the emulated memory image, respectively.

35. (Cancelled)

36. (Previously presented) The computer-readable storage medium of claim 34, further comprising performing the step (a) prior to performing the step (b).

37. (Previously presented) The computer-readable storage medium of claim 36, wherein:

the step (a) further comprises providing an emulated image of the subject processor;

the step (b) further comprises providing a native image of the subject processor following the native execution of the program code; and

the step (c) further comprises comparing the emulated image of the subject processor against the native image of the subject processor.

38. (Original) The computer-readable storage medium of claim 37, wherein the step (a) comprises providing the emulated image of the memory in a load/store buffer associated with the memory, such that the memory is not affected by executing the subject code through the emulator.

39. (Original) The computer-readable storage medium of claim 38, wherein the emulated image of the subject processor includes an image of one or more registers.

40. (Original) The computer-readable storage medium of claim 39, wherein the emulated image of the subject processor includes an image of one or more condition code flags.

41. (Cancelled)

42. (Cancelled)

43. (Previously presented) The computer-readable storage medium of claim 34, further comprising selectively switching between the emulation context for running the emulator on the

subject processor, a target execution context for executing target code produced by the emulator on the subject processor, and the subject native context where the subject code runs natively in the subject processor.

44. (Original) The computer-readable storage medium of claim 43, wherein both the native context and the emulation context employ a single image of the subject code.

45. (Cancelled)

46. (Previously presented) The computer-readable storage medium of claim 34, further comprising selecting between two or more verification modes, and dividing the subject code into the plurality of blocks according to the selected verification mode.

47. (Previously presented) The computer-readable storage medium of claim 46, further comprising repeating the executing and comparing steps for each of the plurality of blocks.

48. (Previously presented) The computer-readable storage medium of claim 47, wherein in a first verification mode each block comprises a single instruction of subject code; in a second verification mode each block comprises a basic block comprising a sequence of instructions from a unique entry instruction to a unique exit instruction; and in a third verification mode each block comprises a group block comprising a plurality of the basic blocks

49. (Previously presented) The computer-readable storage medium of claim 34, further comprising the steps of:

dividing a large segment of the subject code into a plurality of smaller blocks, each block containing one or more instructions from the large segment of subject code; and

performing a verification comparison at a block boundary between each pair of consecutive neighbouring blocks in the plurality of blocks.

50. (Previously presented) The computer-readable storage medium of claim 49, the method further comprising the steps of:

providing the subject processor in the emulation context, where control of the processor rests with the emulator, performing program code conversion on a current block BBn to produce a corresponding block of converted target code, and patching an immediately preceding block of subject code BBn-1 with a return jump;

executing a context switch routine to enter the native context, and executing the immediately preceding block of subject code BBn-1 natively by the subject processor, such that the executing step terminates with the return jump;

executing a context switch routine to return to the emulation context, and performing the verification comparison by comparing a native machine state representing the subject processor following execution of the immediately preceding block BBn-1 with an emulated machine state representing a virtual model of the subject processor held by the emulator following execution of the immediately preceding block BBn-1;

executing a context switch to a target execution context, and modelling execution of the target code corresponding to the current block of subject code BBn in the virtual model of the subject processor held by the emulator, thereby leaving the virtual model in a machine state representing the end of the current block BBn;

and repeating the above steps for each subsequent block in the plurality of blocks, unless the verification comparison reveals an error in the program code conversion.

51. (Original) The computer-readable storage medium of claim 50, the method further comprising restoring the immediately preceding block BBn-1 to remove the return jump.

52. (Original) The computer-readable storage medium of claim 34, the method further comprising the steps of:

selecting a block of the subject code;

executing the block of subject code on the subject processor through the emulator; and

appending a return jump to the block of subject code, and executing the block of subject code natively on the subject processor terminating with the return jump, such that the return jump returns control of the processor to the emulator.

53. (Currently amended) A computer-readable storage medium having emulator software resident thereon in the form of computer readable code executable by a computer to verify program code conversion performed by an emulator by performing the steps of:

dividing a subject code into a plurality of blocks and performing program code conversion to convert one of the plurality of blocks of subject code into target code through an emulator running in a process image on a subject processor according to an emulation context and executing the target code to provide an emulated machine state including selectively inhibiting access by the emulator to a memory associated with the subject processor by buffering load and store requests from the subject processor to the memory in a load/store buffer;

switching to a native context and executing the same one block of subject code directly in the same process image on the same subject processor to provide a native machine state that is stored in the memory associated with the same subject processor; and

comparing the emulated machine state contained in the load/store buffer against the native machine state contained in the memory to verify the program code conversion.

54. (Cancelled)

55. (Previously presented) The computer-readable storage medium of claim 53, the method further comprising selectively inhibiting access to the memory when executing the target code, such that an emulated memory image is provided in the load/store buffer.

56. (Previously presented) The computer-readable storage medium of claim 53, wherein once the program code conversion performed by the emulator running on the subject processor has been verified, and further comprising the step of: comparing execution of the subject code through the

emulator running on the subject processor against execution of the subject code through a second emulator running on a target processor.

57. (Previously presented) The computer-readable storage medium of claim 56, further comprising providing a first host processor as the subject processor, and providing a second host processor as the target processor.

58. (Original) The computer-readable storage medium of claim 57, wherein the subject code is natively executable on the subject processor whilst not being natively executable on the target processor.

59. (Currently amended) A computer-readable storage medium having emulator software resident thereon in the form of computer readable code executable by a computer to verify program code conversion performed by an emulator by performing the steps of:

first dividing a subject code into a plurality of blocks and comparing execution of one subject code block natively on a subject processor against execution of the same subject code block on the same subject processor through a first emulator both in a single process image, thereby verifying program code conversion performed by the first emulator; and

also comparing execution of same subject code block through the first emulator running on the subject processor against execution of the same subject code block through a second emulator running on a target processor, thereby verifying program code conversion performed by the second emulator using the verified program code conversion performed by the first emulator.

60. (Previously presented) The computer-readable storage medium of claim 59, further comprising the steps of:

performing a first program code conversion of the subject code including providing a first virtual model of the subject processor in the first emulator, and comparing the first virtual model against the subject processor; and

performing a second program code conversion of the subject code including providing a second virtual model of the subject processor in the second emulator, and comparing the first virtual model in the first emulator against the second virtual model in the second emulator.

61. (Previously presented) The computer-readable storage medium of claim 60, further comprising providing a single way communication from the first emulator to the second emulator.

62. (Previously presented) The computer-readable storage medium of claim 60, further comprising the steps of:

synchronising the first and second virtual models by sending initial state information from the first emulator to the second emulator;

for each block of subject code, executing the block of subject code through the first emulator and providing a set of subject machine state data and non-deterministic values to the second emulator;

executing the block of subject code in the second emulator substituting the non-deterministic values and providing a set of target machine state data; and

comparing the subject machine state data against the target machine state data and reporting an error if a divergence is detected, otherwise repeating the process for a next block of subject code.

63. (Currently amended) A computer-readable storage medium having emulator software resident thereon in the form of computer readable code executable by a computer to verify program code conversion performed by an emulator by performing the steps of:

(a) dividing a subject code into a plurality of blocks, wherein each block includes at least one instruction;

(b) executing one of the blocks of subject code in a process image on a subject processor through a first emulator according to an emulation context;

(c) switching to a native context, executing the one block of subject code natively on the same subject processor in the same process image and comparing the native execution against the execution of the same one block of subject code on the same subject processor through the first

emulator, thereby verifying program code conversion of the block of subject code performed by the first emulator;

(d) comparing execution of the same one block of subject code through a second emulator running on a target processor against the already verified execution of the same one block of subject code through the first emulator running on the subject processor, thereby verifying program code conversion of the one block of subject code performed by the second emulator; and

(e) repeating steps (b)-(d) for each of the plurality of blocks of the subject code until program code conversion performed by the second emulator is verified for each of the plurality of blocks of the subject code.

64. (Previously presented) The computer-readable storage medium of claim 63, wherein the dividing step comprises dividing the subject code such that each of the plurality of blocks of subject code contains a single instruction.

65. (Previously presented) The computer-readable storage medium of claim 64, wherein after program code conversion performed by the second emulator is verified for each of the plurality of blocks of subject code containing a single instruction, the method further comprises:

secondly repeating the step (a) by redividing the subject code into a plurality of new blocks, wherein each new block is a basic block comprising a sequence of instructions from a unique entry instruction to a unique exit instruction; and

repeating steps (b)-(e) for each basic block, thereby verifying program code conversion performed by the second emulator for every basic block of subject code.

66. (Previously presented) The computer-readable storage medium of claim 65, wherein after program code conversion performed by the second emulator is verified for each of the basic blocks of subject code, the method further comprises:

thirdly repeating the step (a) by redividing the subject code into a plurality of group blocks, wherein each group block comprises a plurality of basic blocks; and

repeating the steps (b)-(e) for each group block, thereby verifying program code conversion performed by the second emulator for every group block of subject code.

67. (Currently amended) An emulator apparatus comprising in combination:

a memory;

a processor; and

emulator code for performing a method of verifying program code conversion, said emulator code comprising code executable by said processor for performing the following steps:

a) dividing a subject code into a plurality of blocks and executing one of the blocks of subject code through an emulator in a process image on the processor as a subject processor according to an emulation context up until a comparable point in the subject code to provide an emulated machine state;

b) performing a context switch to a native context and executing the same block of subject code natively in the same process image on the same subject processor up until the same comparable point in the subject code to provide a native machine state; and

c) comparing the native machine state from execution of the one block of subject code natively on the subject processor against the emulated machine state from execution of the same block of subject code on the same subject processor through the emulator at the comparable point in the subject code;

wherein the native machine state includes a native memory image and the emulated machine state includes an emulated memory image and the step (a) and/or the step (b) includes selectively isolating access to the memory associated with the subject processor to obtain the native memory image and/or the emulated memory image, respectively.

68. (Cancelled)

69. (Original) The emulator apparatus of claim 68, comprising performing the step (a) prior to performing the step (b).

70. (Previously presented) The emulator apparatus of claim 69, wherein:

the step (a) further comprises providing an emulated image of the subject processor;

the step (b) further comprises providing a native image of the subject processor following the native execution of the program code; and

the step (c) further comprises comparing the emulated image of the subject processor against the native image of the subject processor.

71. (Original) The emulator apparatus of claim 70, wherein the step (a) comprises providing the emulated image of the memory in a load/store buffer associated with the memory, such that the memory is not affected by executing the subject code through the emulator.

72. (Original) The emulator apparatus of claim 71, wherein the emulated image of the subject processor includes an image of one or more registers.

73. (Original) The emulator apparatus of claim 72, wherein the emulated image of the subject processor includes an image of one or more condition code flags.

74. (Cancelled)

75. (Cancelled)

76. (Previously presented) The emulator apparatus of claim 67, said emulator code further comprising code executable by said processor for selectively switching between the emulation context for running the emulator on the subject processor, a target execution context for executing

target code produced by the emulator on the subject processor, and the native context where the subject code runs natively in the subject processor.

77. (Original) The emulator apparatus of claim 76, wherein both the native context and the emulation context employ a single image of the subject code.

78. (Cancelled)

79. (Previously presented) The emulator apparatus of claim 67, said emulator code further comprising code executable by said processor for selecting between two or more verification modes, and dividing the subject code into the plurality of blocks according to the selected verification mode.

80. (Previously presented) The emulator apparatus of claim 79, said emulator code further comprising code executable by said processor for repeating the executing and comparing steps for each of the plurality of blocks.

81. (Previously presented) The emulator apparatus of claim 80, wherein in a first verification mode each block comprises a single instruction of subject code; in a second verification mode each block comprises a basic block comprising a sequence of instructions from a unique entry instruction to a unique exit instruction; and in a third verification mode each block comprises a group block comprising a plurality of the basic blocks.

82. (Original) The emulator apparatus of claim 67, said emulator code further comprising code executable by said processor for performing the steps of:

dividing a large segment of the subject code into a plurality of smaller blocks, each block containing one or more instructions from the large segment of subject code; and

performing a verification comparison at a block boundary between each pair of consecutive neighbouring blocks in the plurality of blocks.

83. (Previously presented) The emulator apparatus of claim 82, said emulator code further comprising code executable by said processor for performing the steps of:

providing the subject processor in the emulation context, where control of the processor rests with the emulator, performing program code conversion on a current block BBn to produce a corresponding block of converted target code, and patching an immediately preceding block of subject code BBn-1 with a return jump;

executing a context switch routine to enter the native context, and executing the immediately preceding block of subject code BBn-1 natively by the subject processor, such that the executing step terminates with the return jump;

executing a context switch routine to return to the emulation context, and performing the verification comparison by comparing a native machine state representing the subject processor following execution of the immediately preceding block BBn-1 with an emulated machine state representing a virtual model of the subject processor held by the emulator following execution of the immediately preceding block BBn-1;

executing a context switch to a target execution context, and modelling execution of the target code corresponding to the current block of subject code BBn in the virtual model of the subject processor held by the emulator, thereby leaving the virtual model in a machine state representing the end of the current block BBn; and

repeating the above steps for each subsequent block in the plurality of blocks, unless the verification comparison reveals an error in the program code conversion.

84. (Original) The emulator apparatus of claim 83, said emulator code further comprising code executable by said processor for restoring the immediately preceding block BBn-1 to remove the return jump.

85. (Original) The emulator apparatus of claim 67, said emulator code further comprising code executable by said processor for performing the steps of:

selecting a block of the subject code;

executing the block of subject code on the subject processor through the emulator; and

appending a return jump to the block of subject code, and executing the block of subject code natively on the subject processor terminating with the return jump, such that the return jump returns control of the processor to the emulator.

86. (Currently amended) An emulator apparatus comprising in combination:

a memory;

a processor; and

emulator code for performing a method of verifying program code conversion, said emulator code comprising code executable by said processor for performing the following steps:

dividing a subject code into a plurality of blocks and performing program code conversion to convert one of the plurality of blocks of subject code into target code through an emulator running in a process image on the processor as a subject processor according to an emulation context and executing the target code to provide an emulated machine state including selectively inhibiting access by the emulator to the memory associated with the subject processor by buffering load and store requests from the subject processor to the memory in a load/store buffer;

switching to a native context and executing the same one block of subject code directly in the same process image on the same subject processor to provide a native machine state that is stored in the memory associated with the same subject processor; and

comparing the emulated machine state contained in the load/store buffer against the native machine state contained in the memory to verify the program code conversion.

87. (Cancelled)

88. (Previously presented) The emulator apparatus of claim 86, said emulator code further comprising code executable by said processor for selectively inhibiting access to the memory when executing the target code, such that an emulated memory image is provided in the load/store buffer.

89. (Previously presented) The emulator apparatus of claim 86, wherein after the program code conversion performed by the emulator running on the subject processor has been verified, said emulator code further comprising code executable by said processor for comparing execution of the subject code through the emulator running on the subject processor against execution of the subject code through a second emulator running on a target processor.

90. (Previously presented) The emulator apparatus of claim 89, further comprising a second host processor as the target processor.

91. (Original) The emulator apparatus of claim 90, wherein the subject code is natively executable on the subject processor whilst not being natively executable on the target processor.

92. (Currently amended) An emulator apparatus comprising:

a memory;

a subject processor;

a target processor; and

emulator code for performing a method of verifying program code conversion, said emulator code comprising code executable by said subject processor and said target processor for performing the following steps:

first dividing a subject code into a plurality of blocks and comparing execution of one subject code block natively on the subject processor against execution of the same subject code block on the same subject processor through a first emulator both in a single process image, thereby verifying program code conversion performed by the first emulator; and

also comparing execution of same subject code block through the first emulator running on the subject processor against execution of the same subject code block through a second emulator running on the target processor, thereby verifying program code conversion performed by the second emulator using the verified program code conversion performed by the first emulator.

93. (Original) The emulator apparatus of claim 92, said emulator code further comprising code executable by said processor for performing the following steps:

performing a first program code conversion of the subject code including providing a first virtual model of the subject processor in the first emulator, and comparing the first virtual model against the subject processor; and

performing a second program code conversion of the subject code including providing a second virtual model of the subject processor in the second emulator, and comparing the first virtual model in the first emulator against the second virtual model in the second emulator.

94. (Original) The emulator apparatus of claim 93, said emulator code further comprising code executable by said processor for providing a single way communication from the first emulator to the second emulator.

95. (Previously presented) The emulator apparatus of claim 93, said emulator code further comprising code executable by said processor for performing the following steps:

synchronising the first and second virtual models by sending initial state information from the first emulator to the second emulator;

for each block of subject code, executing the block of subject code through the first emulator and providing a set of subject machine state data and non-deterministic values to the second emulator;

executing the block of subject code in the second emulator substituting the non-deterministic values and providing a set of target machine state data; and

comparing the subject machine state data against the target machine state data and reporting an error if a divergence is detected, otherwise repeating the process for a next block of subject code.

96. (Currently amended) An emulator apparatus comprising:

a memory;

a subject processor;

a target processor; and

emulator code for performing a method of verifying program code conversion, said emulator code comprising code executable by said subject processor and said target processor for performing the following steps:

(a) dividing a subject code into a plurality of blocks, wherein each block includes at least one instruction;

(b) executing one of the blocks of subject code in a process image on the subject processor through a first emulator according to an emulation context;

(c) switching to a native context, executing the one block of subject code natively on the same subject processor in the same process image and comparing the native execution against the execution of the same one block of subject code on the same subject processor through the first emulator, thereby verifying program code conversion of the block of subject code performed by the first emulator;

(d) comparing execution of the same one block of subject code through a second emulator running on the target processor against the already verified execution of the same one block of subject code through the first emulator running on the subject processor, thereby verifying program code conversion of the one block of subject code performed by the second emulator; and

(e) repeating steps (b)-(d) for each of the plurality of blocks of the subject code until program code conversion performed by the second emulator is verified for each of the plurality of blocks of the subject code.

97. (Previously presented) The emulator apparatus of claim 96, wherein the dividing step comprises dividing the subject code such that each of the plurality of blocks of subject code contains a single instruction.

98. (Previously presented) The emulator apparatus of claim 97, wherein after program code conversion performed by the second emulator is verified for each of the plurality of blocks of subject code containing a single instruction, the emulator code further performs the steps of:

secondly repeating the step (a) by redividing the subject code into a plurality of new blocks, wherein each new block is a basic block comprising a sequence of instructions from a unique entry instruction to a unique exit instruction; and

repeating steps (b)-(e) for each basic block, thereby verifying program code conversion performed by the second emulator for every basic block of subject code.

99. (Previously presented) The emulator apparatus of claim 98, wherein after program code conversion performed by the second emulator is verified for each of the basic blocks of subject code, said emulator code further performs the steps of:

thirdly repeating the step (a) by redividing the subject code into a plurality of group blocks, wherein each group block comprises a plurality of basic blocks; and

repeating the steps (b)-(e) for each group block, thereby verifying program code conversion performed by the second emulator for every group block of subject code.